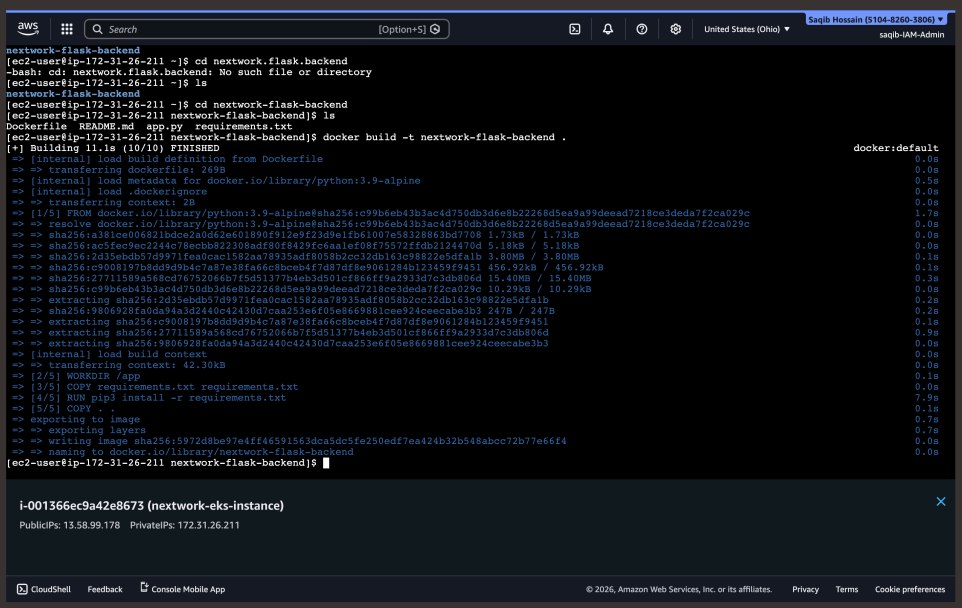
nextwork.org

Set Up Kubernetes Deployment

saqibh49@gmail.com




saqibh49@gmail.com
NextWork Student

Introducing Today's Project!

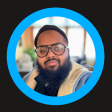
In this project, I will clone a backend app from GitHub, build a Docker image of it, and push that image to Amazon ECR. I'm doing this project because containerizing applications and storing them in a registry is a key part of deploying apps on Kubernetes, and I need to understand this workflow if I want to work with production environments.

Tools and concepts

I used Amazon EKS, EC2, IAM, CloudFormation, Git, Docker, and Amazon ECR to containerize a backend application and prepare it for Kubernetes deployment. Key steps include launching an EC2 instance, setting up eksctl, creating an EKS cluster, cloning the backend code from GitHub, building a Docker image, troubleshooting permission errors, and pushing the image to ECR for my cluster to use.

Project reflection

This project took me approximately 2 hours The most challenging part was solving for all the errors when setting up docker. My favourite part was seeing the backend and understanding how everything comes together to create a flask app.



saqibh49@gmail.com
NextWork Student

nextwork.org

Something new that I learnt from this experience was how many tools work together just to get an app ready for Kubernetes. It's not just about writing code — you need Docker to package it, ECR to store it, EKS to run it, and IAM to make sure everything has the right permissions along the way.

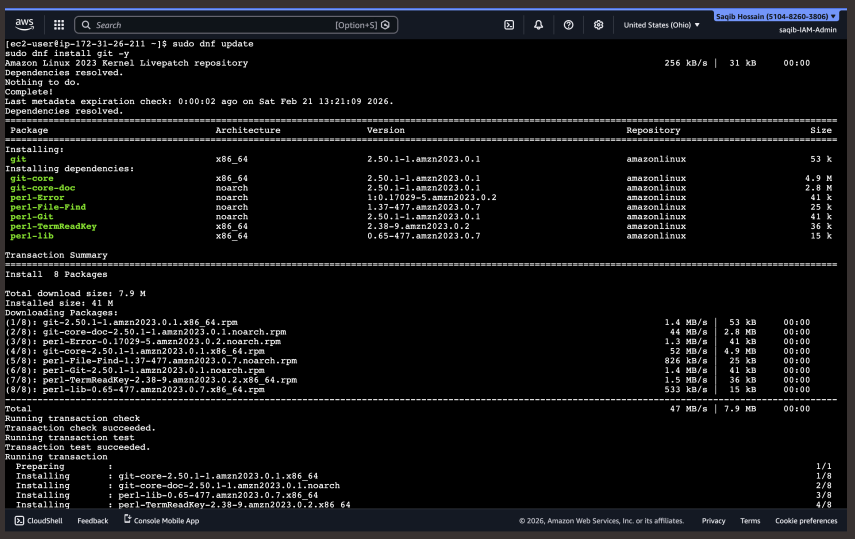
What I'm deploying

To set up today's project, I launched a Kubernetes cluster. Steps I took to do this included launching an EC2 instance, connecting to it, installing eksctl, attaching an IAM role with the right permissions, and running the eksctl create cluster command to spin up my EKS cluster with its node group.

I'm deploying an app's backend

Next, I retrieved the backend that I plan to deploy. An app's backend means the server-side code that handles the logic, data processing, and database interactions that users don't see directly. I retrieved the backend code by installing Git on my EC2 instance using sudo dnf install git -y, then running git clone to download the nextwork-flask-backend repository from GitHub directly onto my instance.

saqibh49@gmail.com
NextWork Student



saqibh49@gmail.com
NextWork Student

Building a container image

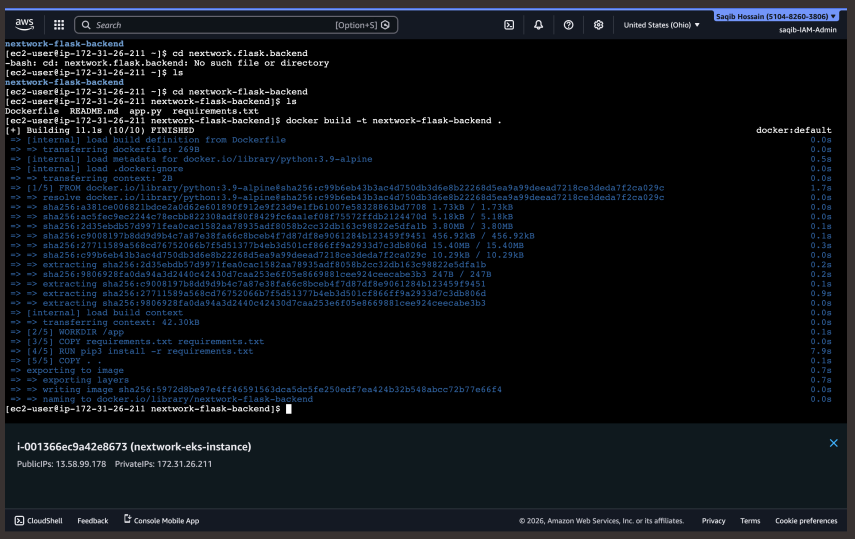
Once I cloned the backend code, my next step is to build a container image of the backend. This is because a container image packages up the app and everything it needs to run into one unit, making it easy to deploy consistently across any environment — especially on my Kubernetes cluster where it'll actually be running.

When I tried to build a Docker image of the backend, I ran into a permissions error because my user didn't have permission to access Docker. Basically, Docker needs root-level access to run, and my ec2-user wasn't part of the Docker group yet, so it got denied.

To solve the permissions error, I added ec2-user to the docker group. The Docker group is basically a list of users that are allowed to communicate with Docker without needing root access. Once I added my user to the group, I could run Docker commands without getting permission denied.

saqibh49@gmail.com
NextWork Student

nextwork.org

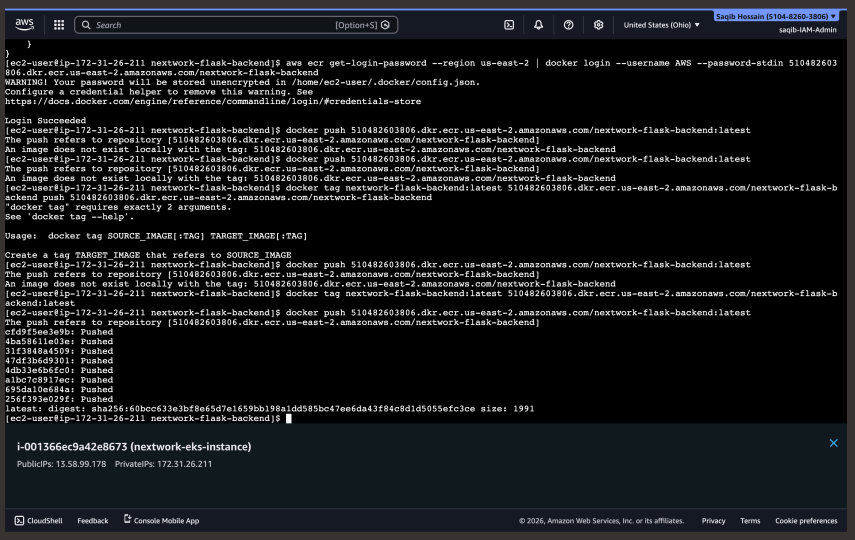


saqibh49@gmail.com
NextWork Student

Container Registry

I'm using Amazon ECR in this project to store my Docker image so my Kubernetes cluster can pull it when it needs to run the app. ECR is a good choice for the job because it's fully managed by AWS, integrates seamlessly with EKS, and keeps my images private and secure without me having to set up my own registry.

Container registries like Amazon ECR are great for Kubernetes deployment because they give your cluster a central place to pull images from. Instead of manually copying images to each node, Kubernetes just pulls the image from the registry whenever it needs to spin up a new container — making deployments faster, more consistent, and easier to manage.




saqibh49@gmail.com
NextWork Student

EXTRA: Backend Explained

After reviewing the app's backend code, I've learnt that this is a pretty straightforward API that pulls data from an external source and reformats it for the frontend. It helped me understand how Flask apps are structured, how APIs call other APIs, and how the backend handles things like filtering data and setting up CORS — all stuff I've dealt with in previous projects but now I can see it directly in the code.

Unpacking three key backend files

The requirements.txt file lists all the Python libraries the app needs to run — like Flask for the web framework, flask-restx for building the API, requests for making HTTP calls, and werkzeug for handling the underlying server utilities. When the Docker image is built, it reads this file and installs everything automatically so the app has all its dependencies ready to go.

The Dockerfile gives Docker instructions on how to build the container image step by step. Key commands in this Dockerfile include FROM, which sets the base image to a lightweight Python environment, WORKDIR, which sets the working directory inside the container, COPY, which moves the app files into the container, RUN, which installs all the dependencies from requirements.txt, and CMD, which tells the container to run the app when it starts up.

The app.py file contains the actual backend application code. It sets up a Flask API with one endpoint — /contents/<topic> — that takes a topic, searches the Hacker News API for articles matching that topic, and returns a list of results with each article's ID, title, and URL. It also includes CORS headers so any frontend can make requests to it, and runs the app on port 8080.

nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects